



Vue.js 2

Wprowadzenie
dla profesjonalistów

Adam Freeman

Tytuł oryginału: Pro Vue.js 2

Tłumaczenie: Krzysztof Rychlicki-Kicior

ISBN: 978-83-283-5498-2

Original edition copyright © 2018 by Adam Freeman.
All rights reserved.

Polish edition copyright © 2019 by HELION SA.
All rights reserved.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Helion SA dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Helion SA nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock Images LLC.

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/vue2wp.zip>

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/vue2wp>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

	O autorze	15
	O korektorze merytorycznym	17
Część I	Zaczynamy pracę z Vue.js	19
Rozdział 1.	Twoja pierwsza aplikacja w Vue.js	21
	Przygotowanie środowiska programistycznego	21
	Instalowanie Node.js	21
	Instalowanie pakietu @vue/cli	22
	Instalowanie narzędzia Git	23
	Instalowanie edytora	23
	Instalowanie przeglądarki	24
	Tworzenie projektu	24
	Struktura podkatalogów w projekcie	24
	Uruchamianie narzędzi deweloperskich	25
	Zamiana treści zastępczych	26
	Dodawanie frameworka do obsługi stylów CSS	28
	Stylowanie elementów HTML	29
	Dodawanie treści dynamicznych	29
	Wyświetlanie listy zadań	31
	Dodawanie przycisku wyboru (checkbox)	33
	Filtrowanie zakończonych zadań	34
	Tworzenie nowych zadań	36
	Trwałe przechowywanie danych	38
	Ostatnie szlify	40
	Podsumowanie	42

Rozdział 2. Zrozumieć Vue.js	43
Czy warto korzystać z Vue.js?	44
Zasada działania aplikacji wielostronicowych	44
Zasada działania SPA	44
Złożoność aplikacji	46
Co muszę wiedzieć?	46
Jak skonfigurować swoje środowisko programistyczne?	46
Jaki jest układ treści w tej książce?	46
Część I. Zaczynamy pracę z Vue.js	47
Część II. Vue.js pod lupą	47
Część III. Zaawansowane funkcje Vue.js	47
Czy znajdę tu dużo przykładów?	47
Gdzie znajdę przykładowe kody?	49
Podsumowanie	49
Rozdział 3. Podstawy HTML i CSS	51
Przygotowania do rozdziału	51
Jak działają elementy języka HTML?	53
Element a jego treść	54
Jak działają atrybuty?	55
Analiza przykładowego dokumentu HTML	56
Jak działa Bootstrap?	58
Stosowanie podstawowych klas Bootstrapa	58
Stosowanie Bootstrapa do tworzenia siatki	60
Stosowanie Bootstrapa do stylowania tabel	60
Stosowanie Bootstrapa do stylowania formularzy	62
Podsumowanie	63
Rozdział 4. Elementarz JavaScriptu	65
Przygotowania do rozdziału	66
Stosowanie instrukcji	68
Tworzenie i używanie funkcji	68
Definicja funkcji z parametrami	70
Tworzenie funkcji zwracających wyniki	71
Przekazywanie funkcji przez argument	71
Zmienne i typy	72
Typy prymitywne	74
Operatory języka JavaScript	76
Instrukcje warunkowe	77
Operator równości a operator identyczności	77
Jawna konwersja typów	78
Obsługa tablic	79
Literały tablicowe	80
Odczyt i modyfikacja zawartości tablicy	80
Przeglądanie zawartości tablicy	81
Operator rozwinięcia	81
Wbudowane metody do obsługi tablic	82

Obsługa obiektów	82
Literały obiektowe	84
Stosowanie funkcji jako metod	85
Kopiowanie właściwości pomiędzy obiektami	85
Moduły w języku JavaScript	86
Tworzenie i używanie modułów	86
Tworzenie wielu mechanizmów w jednym module	88
Łączenie wielu plików w jeden moduł	89
Zasady działania obietnic	90
Problemy z asynchronicznym wykonywaniem operacji	91
Przykład z użyciem obietnic	91
Uproszczenie kodu asynchronicznego	92
Podsumowanie	93
Rozdział 5. Sklep sportowy: prawdziwa aplikacja	95
Tworzenie projektu Sklep sportowy	95
Dodawanie dodatkowych pakietów	96
Przygotowanie REST-owej usługi sieciowej	98
Uruchamianie narzędzi projektowych	100
Tworzenie magazynu danych	101
Tworzenie magazynu produktów	103
Tworzenie listy produktów	104
Dodawanie listy produktów do aplikacji	106
Przetwarzanie cen	106
Obsługa stronicowania listy produktów	108
Obsługa wyboru kategorii	114
Zastosowanie REST-owej usługi sieciowej	117
Podsumowanie	119
Rozdział 6. Sklep sportowy: rozliczenie i zamówienia	121
Przygotowania do rozdziału	121
Tworzenie zastępczej treści dla koszyka	121
Konfiguracja trasowania adresów URL	122
Wyświetlanie trasowanego komponentu	123
Implementacja funkcji koszyka	124
Dodatkowy moduł w magazynie danych	125
Obsługa mechanizmu wyboru produktów	126
Wyświetlanie zawartości koszyka	128
Tworzenie globalnego filtra	131
Testowanie podstawowych funkcji koszyka	132
Utrwalanie koszyka	132
Dodawanie widżetu podsumowania koszyka	135
Obsługa rozliczenia i dodawania zamówień	137
Tworzenie i rejestracja komponentów rozliczenia	138
Dodawanie formularza walidacji	141
Dodawanie pozostałych pól i walidacji	144
Podsumowanie	147

Rozdział 7. Sklep sportowy: skalowanie i administracja	149
Przygotowania do rozdziału	149
Obsługa dużej ilości danych	150
Usprawnienie stronicowania	151
Ograniczanie ilości danych pobieranych przez aplikację	152
Obsługa wyszukiwania	157
Praca nad funkcjami administracyjnymi	161
Implementacja uwierzytelniania	161
Dodawanie struktury komponentu administracyjnego	167
Implementacja zarządzania zamówieniami	169
Podsumowanie	172
Rozdział 8. Sklep sportowy: administrowanie i wdrożenie	173
Przygotowania do rozdziału	173
Dodawanie funkcji administracyjnych	173
Przedstawianie listy produktów	175
Dodawanie treści zastępczej edytora i tras URL	177
Implementacja edytora produktów	178
Wdrażanie sklepu sportowego	181
Przygotowanie aplikacji do wdrożenia	181
Budowanie aplikacji do wdrożenia	185
Testowanie aplikacji gotowej do wdrożenia	186
Wdrożenie aplikacji	188
Podsumowanie	190
Część II Vue.js pod lupą	191
Rozdział 9. Jak działa Vue.js?	193
Przygotowania do rozdziału	193
Dodawanie frameworka Bootstrap CSS	194
Uruchamianie przykładowej aplikacji	194
Tworzenie aplikacji za pomocą API modelu DOM	195
Jak działa aplikacja w modelu DOM?	196
Tworzenie obiektu Vue	198
Stosowanie obiektu Vue	199
Dodawanie funkcji obsługi zdarzenia	200
Modyfikacja komunikatu	201
Zasada działania obiektu Vue	202
Komponenty w praktyce	203
Rejestracja i wdrażanie komponentu	204
Oddzielanie szablonu od kodu JavaScript	205
Podsumowanie	207
Rozdział 10. Projekty i narzędzia Vue.js	209
Tworzenie projektu aplikacji Vue.js	209
Konfiguracja lintera	212
Zakończenie konfiguracji projektu	212
Omówienie struktury projektu	213
Omówienie katalogu z kodem źródłowym	214
Omówienie katalogu pakietów	216

Omówienie narzędzi deweloperskich	218
Omówienie procesów kompilacji i transformacji	219
Omówienie serwera deweloperskiego HTTP	221
Omówienie mechanizmu zamiany modułów na gorąco	222
Omówienie wyświetlania błędów	224
Stosowanie lintera	226
Dostosowywanie reguł lintera	229
Debugowanie aplikacji	231
Analiza stanu aplikacji	231
Omówienie debuggera w przeglądarce	231
Konfiguracja narzędzi deweloperskich	233
Budowanie aplikacji do wdrożenia	233
Instalacja i zastosowanie serwera HTTP	236
Podsumowanie	237
Rozdział 11. Omówienie wiązań danych	239
Przygotowania do tego rozdziału	240
Omówienie składników komponentu	242
Omówienie elementu template	242
Omówienie elementu script	243
Omówienie elementu style	243
Zmiany komponentu w przykładowej aplikacji	243
Wyświetlanie wartości danych	244
Stosowanie złożonych wyrażeń w wiązaniach danych	247
Przeliczanie wartości we właściwościach obliczanych	249
Obliczanie wartości danych za pomocą metody	252
Formatowanie wartości danych za pomocą filtrów	255
Podsumowanie	260
Rozdział 12. Stosowanie podstawowych dyrektyw	261
Przygotowania do tego rozdziału	262
Ustawianie zawartości tekstowej elementu	263
Wyświetlanie czystego kodu HTML	265
Wyświetlanie wybranych elementów	267
Wyświetlanie wybranych elementów sąsiednich	268
Wybór fragmentów zawartości	270
Wybór wyświetlanych elementów za pomocą stylów CSS	272
Ustawianie atrybutów i właściwości elementu	274
Stosowanie obiektu do konfiguracji klas	276
Ustawianie pojedynczych stylów	277
Ustawianie innych atrybutów	279
Ustawianie wielu atrybutów	280
Ustawianie właściwości HTML:Element	281
Podsumowanie	283
Rozdział 13. Obsługa dyrektywy Repeater	285
Przygotowania do tego rozdziału	285
Przeglądanie tablicy	287
Stosowanie aliasu	289
Określanie klucza	291

Pobieranie indeksu elementu	293
Wykrywanie zmian w tablicy	296
Wyliczanie właściwości obiektu	298
Właściwości obiektu a kwestia kolejności	300
Powtarzanie elementów HTML bez źródła danych	302
Stosowanie właściwości obliczanych z dyrektywą v-for	303
Stronicowanie danych	303
Filtrowanie i sortowanie danych	305
Podsumowanie	307
Rozdział 14. Obsługa zdarzeń	309
Przygotowania do tego rozdziału	309
Obsługa zdarzeń	311
Omówienie zdarzeń i obiektów zdarzeń	312
Stosowanie metody do obsługi zdarzeń	313
Połączenie zdarzeń, metod i elementów powtarzanych	315
Nasłuchiwanie wielu zdarzeń z tego samego elementu	317
Stosowanie modyfikatorów obsługi zdarzeń	320
Zarządzanie propagacją zdarzeń	320
Zapobieganie duplikacji zdarzeń	326
Omówienie modyfikatorów zdarzeń myszy	327
Omówienie modyfikatorów zdarzeń klawiatury	328
Podsumowanie	330
Rozdział 15. Obsługa elementów formularzy	331
Przygotowania do tego rozdziału	331
Tworzenie dwukierunkowych wiązań modeli	333
Dodawanie wiązania dwukierunkowego	334
Dodawanie kolejnego elementu wejściowego	335
Upraszczanie wiązań dwukierunkowych	337
Wiązania z elementami formularzy	338
Wiązania do pól tekstowych	338
Wiązania do przycisków opcji i wyboru	339
Wiązania do elementów typu select	341
Stosowanie modyfikatorów dyrektywy v-model	343
Formatowanie wartości jako liczb	343
Opóźnianie aktualizacji	344
Usuwanie białych znaków	345
Wiązania do różnych typów danych	346
Wybór tablicy elementów	346
Stosowanie własnych wartości w elementach formularza	348
Walidacja danych w formularzu	351
Definiowanie reguł walidacji	353
Stosowanie funkcji walidacji	354
Bieżące reagowanie na zmiany	357
Podsumowanie	358

Rozdział 16. Stosowanie komponentów	359
Przygotowania do tego rozdziału	359
Omówienie komponentów jako podstawowych składników aplikacji	361
Omówienie nazw komponentów i elementów dzieci	363
Wykorzystywanie możliwości komponentów w komponentach-dzieciach	365
Omówienie izolacji komponentów	366
Stosowanie propów w komponentach	368
Tworzenie własnych zdarzeń	373
Stosowanie slotów komponentów	376
Podsumowanie	381
 Część III Zaawansowane funkcje Vue.js	 383
Rozdział 17. Omówienie cyklu życia komponentu Vue.js	385
Przygotowania do tego rozdziału	386
Omówienie cyklu życia komponentu	388
Omówienie fazy tworzenia	389
Omówienie fazy montażu	390
Omówienie fazy aktualizacji	392
Omówienie fazy zniszczenia	398
Obsługa błędów komponentów	400
Podsumowanie	403
 Rozdział 18. Luźno powiązane komponenty	405
Przygotowania do tego rozdziału	406
Tworzenie komponentu do wyświetlania produktu	408
Tworzenie komponentu edytora produktu	409
Wyświetlanie komponentów-dzieci	410
Omówienie wstrzykiwania zależności	411
Tworzenie usługi	411
Konsumowanie usługi za pomocą wstrzykiwania zależności	412
Przesłanie usług pochodzących od przodków	413
Tworzenie reaktywnych usług	415
Zaawansowane wstrzykiwanie zależności	417
Stosowanie szyny zdarzeń	420
Wysyłanie zdarzeń za pomocą szyny zdarzeń	420
Odbieranie zdarzeń z szyny zdarzeń	421
Tworzenie lokalnych szyn zdarzeń	424
Podsumowanie	426
 Rozdział 19. Stosowanie REST-owych usług sieciowych	427
Przygotowania do tego rozdziału	427
Przygotowanie serwera HTTP	428
Przygotowanie przykładowej aplikacji	429
Uruchamianie przykładowej aplikacji i serwera HTTP	432
Omówienie REST-owych usług sieciowych	433
Konsumowanie REST-owej usługi sieciowej	435
Obsługa danych odpowiedzi	435
Wykonywanie żądania HTTP	436

Otrzymywanie odpowiedzi	437
Przetwarzanie danych	438
Tworzenie usługi HTTP	440
Konsumowanie usługi HTTP	440
Dodawanie pozostałych operacji HTTP	441
Tworzenie usługi obsługi błędów	444
Podsumowanie	447
Rozdział 20. Stosowanie magazynu danych	449
Przygotowania do tego rozdziału	449
Tworzenie i używanie magazynu danych	452
Omówienie podziału na stan i mutacje	454
Udostępnianie magazynu danych Vuex	456
Stosowanie magazynu danych	456
Analiza zmian w magazynie danych	460
Definiowanie właściwości obliczanych w magazynie danych	461
Stosowanie gettera w komponencie	463
Przekazywanie argumentów do getterów	464
Wykonywanie operacji asynchronicznych	464
Otrzymywanie powiadomień o zmianach	468
Mapowanie funkcji magazynu danych w komponentach	471
Stosowanie modułów magazynu danych	474
Rejestrowanie i stosowanie modułu magazynu danych	475
Stosowanie przestrzeni nazw modułów	478
Podsumowanie	480
Rozdział 21. Komponenty dynamiczne	481
Przygotowania do tego rozdziału	482
Przygotowywanie komponentów do dynamicznego cyklu życia	483
Pobieranie danych aplikacji	483
Zarządzanie zdarzeniami obserwatora	484
Dynamiczne wyświetlanie komponentów	485
Przedstawianie różnych komponentów w elemencie HTML	486
Wybór komponentów za pomocą wiązania danych	486
Automatyczna nawigacja w aplikacji	490
Stosowanie komponentów asynchronicznych	494
Wyłączanie podpowiedzi wstępnego pobierania	497
Konfiguracja leniwego ładowania	498
Podsumowanie	501
Rozdział 22. Trasowanie URL	503
Przygotowania do tego rozdziału	503
Rozpoczynamy pracę z trasowaniem URL	505
Dostęp do konfiguracji trasowania	507
Stosowanie systemu trasowania do wyświetlania komponentów	507
Nawigowanie do innych adresów URL	510
Omówienie i konfiguracja dopasowania tras URL	513
Omówienie dopasowania i formatowania adresów URL	514
Stosowanie API historii HTML5 do trasowania	515

Stosowanie aliasu trasy	518
Pobieranie danych trasowania w komponentach	519
Dynamiczne dopasowywanie tras	522
Stosowanie wyrażeń regularnych do dopasowywania adresów URL	525
Tworzenie tras nazwanych	528
Obsługa zmian w nawigacji	531
Podsumowanie	534
Rozdział 23. Elementy związane z trasowaniem URL	535
Przygotowania do tego rozdziału	536
Obsługa elementów router-link	537
Wybór rodzaju elementu	538
Wybór zdarzenia nawigacji	541
Stylowanie elementów łącza routera	542
Tworzenie tras zagnieżdżonych	546
Planowanie układu aplikacji	547
Dodawanie komponentów do projektu	547
Definiowanie tras	548
Tworzenie elementów nawigacji	550
Testowanie klas zagnieżdżonych	551
Obsługa nazwanych elementów router-view	553
Podsumowanie	557
Rozdział 24. Zaawansowane trasowanie URL	559
Przygotowania do tego rozdziału	559
Stosowanie odrębnych plików dla powiązanych tras	560
Ochrona tras	562
Definiowanie globalnych strażników nawigacji	562
Definiowanie strażników dla konkretnych tras	566
Definiowanie strażników tras dla komponentów	570
Ładowanie komponentów na żądanie	577
Wyświetlanie komponentu z komunikatem ładowania	578
Tworzenie komponentów bez obsługi trasowania	582
Podsumowanie	585
Rozdział 25. Przejścia	587
Przygotowania do tego rozdziału	587
Tworzenie komponentów	589
Konfiguracja trasowania URL	592
Tworzenie elementów nawigacji	592
Rozpoczynamy pracę z przejściami	594
Omówienie klas przejść i przejść CSS	596
Omówienie sekwencji przejścia	597
Stosowanie biblioteki do obsługi animacji	598
Przełączanie pomiędzy wieloma elementami	599
Stosowanie przejścia do elementów z trasowaniem URL	601
Stosowanie przejścia podczas pojawiania się elementu	603
Stosowanie przejść dla zmian w kolekcji	604

Stosowanie zdarzeń przejść	606
Stosowanie zdarzeń początkowych i końcowych	608
Przyciąganie uwagi do innych zmian	609
Podsumowanie	612
Rozdział 26. Rozszerzanie możliwości Vue.js	613
Przygotowania do tego rozdziału	614
Tworzenie własnych dyrektyw	616
Omówienie zasady działania dyrektyw	618
Stosowanie wyrażeń własnych dyrektyw	620
Stosowanie argumentów własnej dyrektywy	621
Stosowanie modyfikatorów własnej dyrektywy	622
Komunikacja między funkcjami haków	624
Dyrektywy jednofunkcyjne	625
Tworzenie domieszek komponentów	626
Tworzenie wtyczki Vue.js	629
Tworzenie wtyczki	632
Stosowanie wtyczki	633
Podsumowanie	635
Skorowidz	637

ROZDZIAŁ 9.



Jak działa Vue.js?

Na początku swojej przygody z Vue.js możesz poczuć się przytłoczony dużą liczbą zagadnień do zrozumienia. Nie wszystko wydaje się mieć sens od samego początku. W tym rozdziale objaśniam, jak działa Vue.js, i pokazuję, że nie ma w nim żadnej magii — warto o tym pamiętać, gdy będę zagłębiał się w tajniki poszczególnych zagadnień Vue.js w kolejnych rozdziałach. Tabela 9.1 podsumowuje bieżący rozdział.

Tabela 9.1. Podsumowanie rozdziału

Problem	Rozwiązanie	Listing
Utwórz prostą aplikację webową bez pomocy Vue.js.	Skorzystaj z API DOM (ang. <i>Document Object Model</i> — obiektowy model dokumentu).	9.6
Utwórz prostą aplikację webową z pomocą Vue.js.	Utwórz obiekt Vue i skonfiguruj go przy użyciu właściwości <code>el</code> i <code>template</code> .	9.7
Zaprezentuj wartość danych użytkownikowi.	Zdefiniuj właściwość danych i użyj wiązania danych.	9.8
Zareaguj na działania użytkownika za pomocą zdarzeń.	Skorzystaj z dyrektywy <code>v-on</code> .	9.9
Wygeneruj wartości danych, które wymagają obliczeń.	Zdefiniuj właściwość obliczaną.	9.10
Utwórz wielokrotnego użytku funkcjonalną jednostkę aplikacji.	Utwórz i zastosuj komponent.	9.11 – 9.13
Zdefiniuj szablon komponentu jako kod HTML.	Skorzystaj z elementu <code>template</code> .	9.14 – 9.16

Przygotowania do rozdziału

Aby utworzyć przykładowy projekt, otwórz wiersz poleceń, przejdź do wybranego katalogu i wykonaj polecenie z listingu 9.1. To polecenie wymaga posiadania narzędzi z rozdziału 1. Musisz wykonać opisane w nim operacje, aby utworzyć projekt.

Listing 9.1. Tworzenie przykładowego projektu

```
vue create nomagic
```

-
- **Wskazówka** Przykładowy projekt do tego rozdziału — podobnie jak do wszystkich innych — można pobrać z serwera FTP wydawnictwa Helion pod adresem <ftp://ftp.helion.pl/przyklady/vue2wp.zip>.
-

Gdy zostaniesz poproszony o wybór ustawień, wybierz opcję default. Projekt zostanie utworzony, a pakiety do wszystkich narzędzi deweloperskich zostaną pobrane i zainstalowane, co może trochę potrwać.

-
- **Uwaga** W trakcie pisania niniejszej książki pakiet `@vue/cli` był dostępny w wersji beta. W związku z możliwymi zmianami warto zapoznać się z erratą dostępną w repozytorium oryginalnego wydania książki pod adresem <https://github.com/Apress/pro-vue-js-2>.
-

Po utworzeniu projektu do katalogu *nomagic* dodaj plik *vue.config.js* o zawartości jak w listingu 9.2. Plik ten jest używany do konfigurowania narzędzi deweloperskich Vue.js, które opisuję w rozdziale 10.

Listing 9.2. Zawartość pliku *nomagic/vue.config.js*

```
module.exports = {  
  runtimeCompiler: true  
}
```

Dodawanie frameworka Bootstrap CSS

Aby dodać pakiet Bootstrap CSS do projektu, wykonaj polecenie z listingu 9.3 w katalogu *nomagic*. Jest to framework CSS używany do stylowania treści HTML w tym rozdziale.

Listing 9.3. Dodawanie pakietu *Bootstrap*

```
npm install bootstrap@4.0.0
```

Po zakończeniu instalacji otwórz plik *src/main.js* i dodaj instrukcje z listingu 9.4.

Listing 9.4. Dodawanie *Bootstrapa* w pliku *src/main.js*

```
import Vue from 'vue'  
import App from './App.vue'  
import "bootstrap/dist/css/bootstrap.min.css";  
Vue.config.productionTip = false  
new Vue({  
  render: h => h(App)  
}).$mount('#app')
```

Ta instrukcja dołączy style CSS Bootstrapa do treści przesyłanych do przeglądarki.

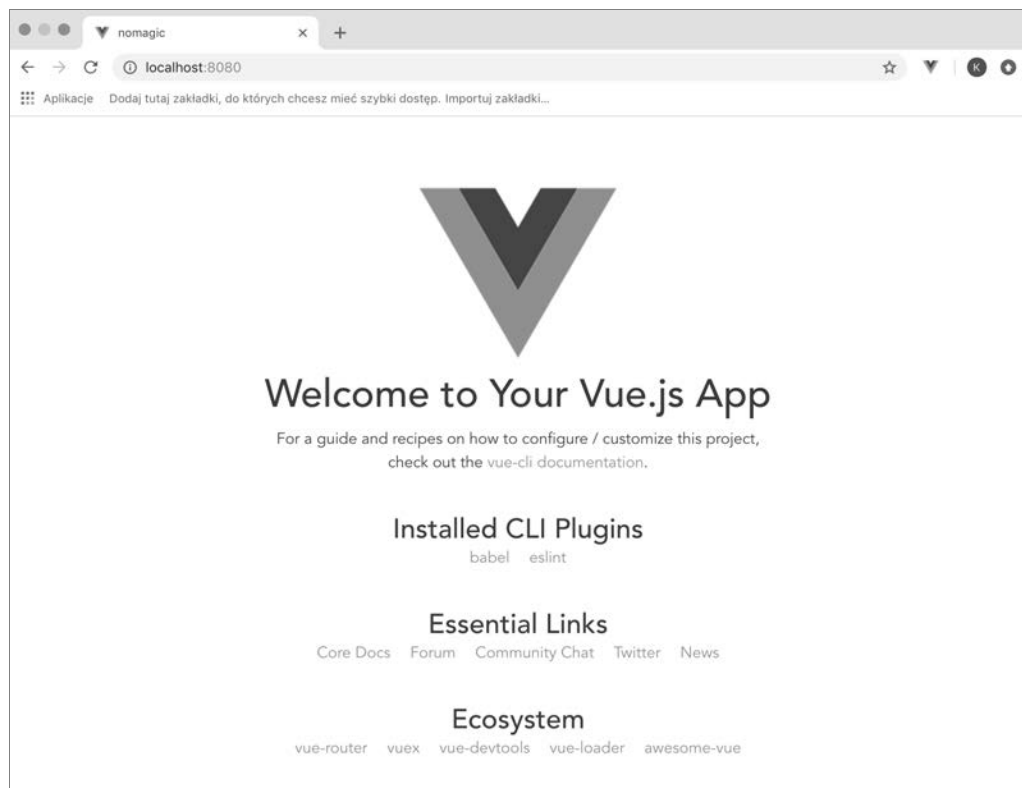
Uruchamianie przykładowej aplikacji

Projekt utworzony za pomocą polecenia z listingu 9.1 łączy narzędzia wymagane w procesie tworzenia aplikacji w Vue.js. Więcej na temat zasad użycia tych narzędzi znajdziesz w rozdziale 10. Na razie, aby uruchomić narzędzia deweloperskie, uruchom polecenie z listingu 9.5 w katalogu *nomagic*.

Listing 9.5. Uruchamianie narzędzi deweloperskich

```
npm run serve
```

W tym momencie nastąpi start serwera deweloperskiego HTTP. Otwórz nowe okno przeglądarki i przejdź pod adres `http://localhost:8080`. Zobaczysz treść zastępczą jak na rysunku 9.1.



Rysunek 9.1. Uruchamianie przykładowej aplikacji

Tworzenie aplikacji za pomocą API modelu DOM

Zacznijmy od utworzenia prostej aplikacji webowej bez użycia choćby odrobiny kodu Vue.js. Następnie pokażę, jak osiągnąć ten sam efekt za pomocą podstawowych opcji Vue.js.

W tym celu utworzę plik `main.js`, który zazwyczaj zawiera kod JavaScript odpowiedzialny za inicjalizację aplikacji Vue.js. Plik `main.js` to zwykły plik języka JavaScript, co oznacza, że możemy usunąć kod odpowiedzialny za konfigurację Vue.js i wstawić tam dowolne inne instrukcje.

W listingu 9.6 zamieniam domyślną treść pliku `main.js` na zwykłe instrukcje języka JavaScript, które korzystają z obiektowego modelu dokumentu (DOM). API modelu DOM oferuje wszystkie przeglądarki w celu udostępniania treści dokumentu HTML w obrębie skryptów języka JavaScript. Model DOM stanowi podstawę działania praktycznie wszystkich aplikacji webowych.

Listing 9.6. Zamiana treści w pliku `src/main.js`

```
require('../node_modules/bootstrap/dist/css/bootstrap.min.css')
let counter = 1;
let container = document.createElement("div");
container.classList.add("text-center", "p-3");
let msg = document.createElement("h1");
msg.classList.add("bg-primary", "text-white", "p-3");
```

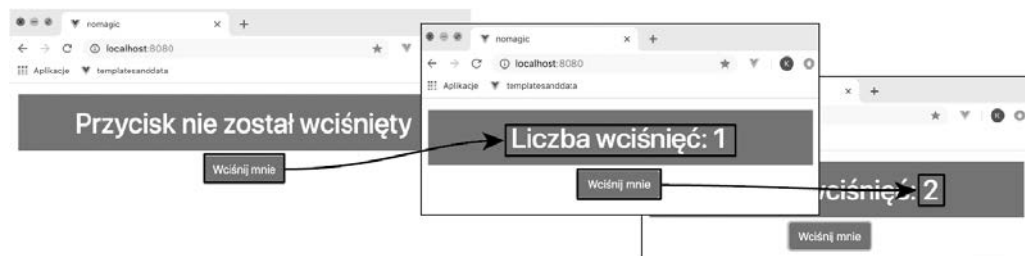
```

msg.textContent = "Przycisk nie został wciśnięty";
let button = document.createElement("button");
button.textContent = "Wciśnij mnie";
button.classList.add("btn", "btn-secondary");
button.onclick = () => msg.textContent = `Liczba wciśnięć: ${counter++}`;
container.appendChild(msg);
container.appendChild(button);
let app = document.getElementById("app");
app.parentElement.replaceChild(container, app);

```

Nie martw się instrukcjami z powyższego kodu. W większości przypadków nie będziesz korzystać z API modelu DOM w projektach Vue.js. Celem tego przykładu jest przedstawienie pliku *main.js* jako zwykłego skryptu, zawierającego zwyczajny kod JavaScript, który może korzystać ze standardowych funkcji udostępnianych przez przeglądarkę.

Narzędzia deweloperskie uruchomione za pomocą polecenia z listingu 9.6 automatycznie wykrywają zmiany w plikach projektu. Zmiany są kompilowane, pakowane w jeden plik zawierający wszystkie funkcje aplikacji i wysyłane z powrotem do przeglądarki. Oznacza to, że po zapisaniu pliku *main.js* przeglądarka zostanie odświeżona, a Ty zobaczysz treść jak na rysunku 9.2. Na początku widać prosty komunikat, ale po kliknięciu przycisku zostaje on zastąpiony licznikiem. Licznik będzie zwiększać się po każdym wciśnięciu przycisku.



Rysunek 9.2. Działanie bezpośrednio na modelu DOM

Jak działa aplikacja w modelu DOM?

Z API modelu DOM korzystam w celu utworzenia elementu `div`, który zawiera elementy `h1` i `button`, w momencie wykonywania pliku *main.js*. Jednocześnie wiążę te elementy z klasami, które odpowiadają stylom zdefiniowanym we frameworku Bootstrap CSS, określającymi wygląd tych elementów. Następnie ustawiam słuchacza zdarzeń, który reaguje na kliknięcie przycisku poprzez modyfikowanie stanu licznika i wyświetlanie komunikatu w elemencie `h1`.

Po wykonaniu wyżej opisanych operacji lokalizuję istniejący w dokumencie element o ID równym `app` i zamieniam go na utworzony element `div`.

```

...
let app = document.getElementById("app");
app.parentElement.replaceChild(container, app);
...

```

Poniżej przedstawiam sposób, w jaki użytkownik otrzyma aplikację. Przeglądarka wysyła żądanie pod adres `http://localhost:8000`, a serwer deweloperski HTTP odpowiada zawartością pliku *index.html*:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">

```



```

    <title>nomagic</title>
  </head>
  <body>
    <div id="app"></div>
  </body>
</html>

```

Zaznaczyłem wyraźnie element zamieniany przez kod w języku JavaScript. To właśnie tu znajdzie się kod HTML generowany w JavaScriptcie.

Gdy serwer wygeneruje odpowiedź na żądanie HTTP przeglądarki, automatycznie zostanie wstawiony element script, który wczyta plik JavaScript zawierający cały kod projektu. Możesz to sprawdzić, klikając prawym przyciskiem myszy w oknie przeglądarki i wybierając z menu kontekstowego opcję *Wyświetl źródło strony*.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <title>nomagic</title>
  </head>
  <body>
    <div id="app"></div>
    <script type="text/javascript" src="/app.js"></script>
  </body>
</html>

```

Dostarczony plik JavaScript zawiera kod z pliku *main.js* i wszelkie jego zależności. Przeglądarka wykonuje instrukcje w pliku JavaScript, co prowadzi do wygenerowania omówionej wcześniej treści.

Kod HTML powstały w wyniku połączenia statycznych fragmentów z pliku *index.html* i dynamicznych znaczników wygenerowanych przez plik *main.js* można podejrzeć poprzez kliknięcie prawym przyciskiem w oknie przeglądarki i wybranie opcji *Zbadaj w menu kontekstowym*. W ten sposób zobaczymy widok na żywo dokumentu, utworzony za pomocą API modelu DOM.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <title>nomagic</title>
  </head>
  <body>
    <div class="text-center p-3">
      <h1 class="bg-primary text-white p-3">Przycisk nie został wciśnięty</h1>
      <button class="btn btn-secondary">Wciśnij mnie</button>
    </div>
    <script type="text/javascript" src="/app.js"></script>
  </body>
</html>

```

Effekt nie jest zbyt imponujący, ale w ten sposób pokazujemy, że zwykły kod JavaScript, z pomocą API modelu DOM, również może być używany do zamiany elementów w dokumentach HTML, a także do tworzenia nowej treści i reagowania na zdarzenia pochodzące od użytkownika. To są podstawy każdej aplikacji webowej — w tym także tej tworzonej w Vue.js.

Tworzenie obiektu Vue

Aplikacje Vue.js również korzystają z API modelu DOM do tworzenia treści HTML, reagowania na zdarzenia i aktualizowania wartości danych. Różnica polega na tym, że podejście z zastosowaniem Vue.js jest bardziej eleganckie, łatwiejsze do zrozumienia i bardziej skalowalne.

Przed chwilą skorzystałem z pliku *main.js* jako wygodnej metody wykonania kodu JavaScript, jednak jego głównym celem w aplikacji Vue.js jest utworzenie obiektu Vue, który stanowi punkt startowy całej aplikacji. W listingu 9.7 zamieniam kod API modelu DOM w pliku *main.js* na zestaw instrukcji, który tworzy obiekt Vue, i przywracam tym samym plikowi *main.js* jego pierwotne przeznaczenie.

Listing 9.7. Tworzenie obiektu Vue w pliku *src/main.js*

```
require('../node_modules/bootstrap/dist/css/bootstrap.min.css')
import Vue from "vue"
new Vue({
  el: "#app",
  template: `<div class="text-center p-3">
    <h1 class="bg-secondary text-white p-3">
      Vue: Przycisk nie został wciśnięty
    </h1>
    <button class="btn btn-secondary">
      Wciśnij mnie
    </button>
  </div>`
});
```

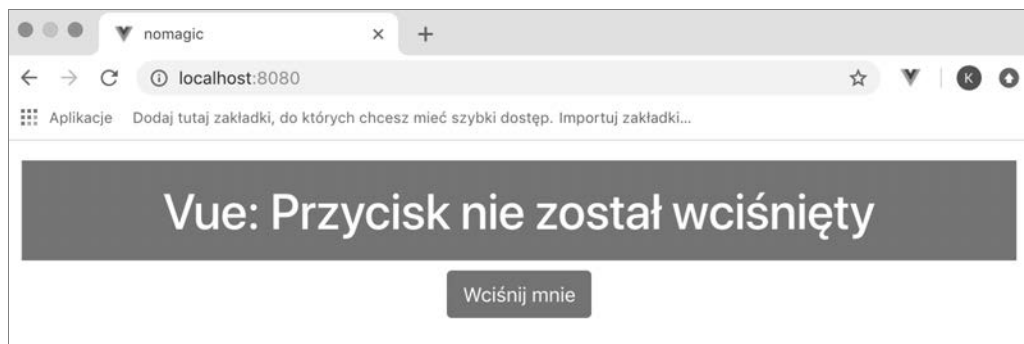
Instrukcja `import` jest niezbędna do zaimportowania obiektu Vue z modułu `vue`, pobranego do katalogu *node_modules* w momencie utworzenia projektu. Obiekt Vue został utworzony za pomocą słowa kluczowego `new`, a konstruktor przyjął obiekt konfiguracji, którego właściwości pozwalają na kontrolę zachowania aplikacji i definiują treści przedstawiane użytkownikowi.

W tym przykładzie mamy do czynienia z dwoma właściwościami konfiguracji: `el` i `template`. Obiekt Vue korzysta z właściwości `el`, aby zidentyfikować element, który zostanie zamieniony przez zawartość aplikacji w pliku *index.html*. Właściwość `template` dostarcza zawartość HTML, która zamieni dotychczasową zawartość elementu ukrytego za właściwością `el`. Jak pewnie pamiętasz, te zadania musiałem z pomocą API modelu DOM wykonywać ręcznie w poprzednim przykładzie.

W listingu 9.7 ustawiam właściwość `el` na wartość `#app`, która pobiera element o `id` równym `app`. Ustawiam właściwość `template` w taki sposób, aby zawierała tę samą strukturę kodu HTML co w listingu 9.6. Tym razem mogę jednak po prostu napisać kod HTML, zamiast tworzyć identyczną konstrukcję za pomocą wywołań kodu JavaScript. Obiekt konfiguracji jest zdefiniowany w kodzie JavaScript, co oznacza, że kod HTML w nim osadzony musi być zawarty jako prawidłowy łańcuch znaków JavaScript. Skorzystałem z grawisa (znaku ```), dzięki czemu jestem w stanie pisać kod w wielu wierszach, przez co jest on łatwiejszy do odczytu.

■ **Ostrzeżenie** Możesz korzystać z właściwości `template` tylko, jeśli przesłoniłeś ustawienia projektu jak w listingu 9.2. Domyślnie funkcja do przetwarzania tekstów zawierających szablony jest wyłączona w Vue.js.

Po zapisaniu zmian w pliku *main.js* przeglądarka zostanie odświeżona, a treść będzie przypominać tę z rysunku 9.3. Zmieniła się treść komunikatu, a także kolor tła.



Rysunek 9.3. Zastosowanie obiektu Vue

Stosowanie obiektu Vue

Mój obiekt Vue wyświetla zawartość HTML użytkownikowi, ale to tylko część funkcji, które są mi potrzebne. Kolejnym krokiem jest dodanie danych i wyświetlenie ich użytkownikowi. W listingu 9.8 dodaję zmienną `counter` i modyfikuję łańcuch `template` tak, aby został wyświetlony użytkownikowi.

Listing 9.8. Dodawanie zmiennej do pliku `src/main.js`

```
require('../node_modules/bootstrap/dist/css/bootstrap.min.css')
import Vue from "vue"
new Vue({
  el: "#app",
  template: `<div class="text-center p-3">
    <h1 class="bg-secondary text-white p-3">
      Liczba wciśnięć: {{ counter }}
    </h1>
    <button class="btn btn-secondary">
      Wciśnij mnie
    </button>
  </div>`,
  data: {
    counter: 0
  }
});
```

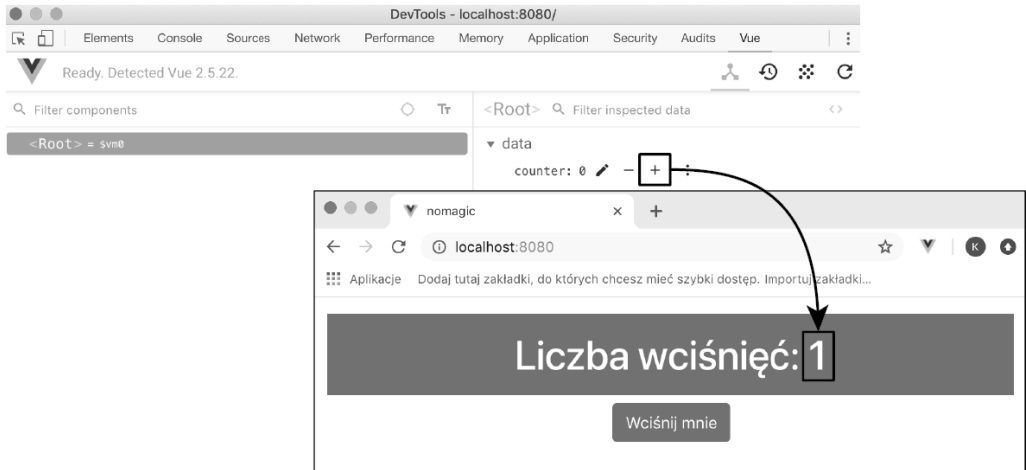
Dodałem właściwość `data` do obiektu konfiguracyjnego aplikacji. Obiekt definiuje właściwość `counter` z wartością początkową równą zero. Aby wyświetlić wartość licznika, stosuję proste wiązanie danych w elemencie `h1`.

```
...
<h1 class="bg-secondary text-white p-3">
  Liczba wciśnięć: {{ counter }}
</h1>
...
```

Wiązania danych to niezwykle ważna funkcja Vue.js, która łączy obiekt `data` z treścią właściwości `template`. Gdy Vue.js wyświetla treść szablonu, poszukuje w nim wiązań danych, rozpoznaje je jako wyrażenia języka JavaScript i dołącza wynik operacji jako kod HTML. W tym przypadku wiązanie danych odwołuje się do jednej z właściwości obiektu `data`. W rezultacie wartość zmiennej `counter` zostanie wstawiona na stronę w elemencie `h1`.

Dane są **reaktywne, odświeżane na żywo** — oznacza to, że zmiana wartości właściwości `counter` zostanie automatycznie odzwierciedlona w elemencie `h1`. Po utworzeniu obiektu `Vue` następuje przetworzenie obiektu `data` i zamiana właściwości, przez co możliwe jest wykrycie zmian w ich wartościach.

Konsekwencję reaktywności danych widać w narzędziu `Vue Devtools`. Otwórz narzędzia deweloperskie za pomocą klawisza `F12`, przejdź na zakładkę `Vue` i kliknij element `<Root>`. W panelu po prawej stronie przesunij kursor na element `counter` i kliknij przycisk `+`, aby zwiększyć jego wartość. `Vue.js` wykryje zmianę wartości właściwości i przetworzy kod szablonu ponownie, generując efekt jak na rysunku 9.4. Szczegóły instalacji tej wtyczki znajdziesz na stronie <https://github.com/vuejs/vue-devtools>.



Rysunek 9.4. Zmiana wartości zmiennej reaktywnej

Dodawanie funkcji obsługi zdarzenia

Kolejnym krokiem w tworzeniu tego przykładu jest zwiększenie wartości zmiennej `counter` automatycznie po kliknięciu przycisku. Poprzednio mogłem powiązać funkcję obsługi zdarzenia bezpośrednio za pomocą instrukcji języka JavaScript. Tym razem zastosuję inne podejście (listing 9.9).

Listing 9.9. Obsługa zdarzenia w pliku `src/main.js`

```
require('../node_modules/bootstrap/dist/css/bootstrap.min.css')
import Vue from "vue"
new Vue({
  el: "#app",
  template: `<div class="text-center p-3">
    <h1 class="bg-secondary text-white p-3">
      Liczba wciśnień: {{ counter }}
    </h1>
    <button class="btn btn-secondary" v-on:click="handleClick">
      Wciśnij mnie
    </button>
  </div>`,
  data: {
    counter: 0
  },
  methods: {
    handleClick() {
      this.counter++;
    }
  }
})
```

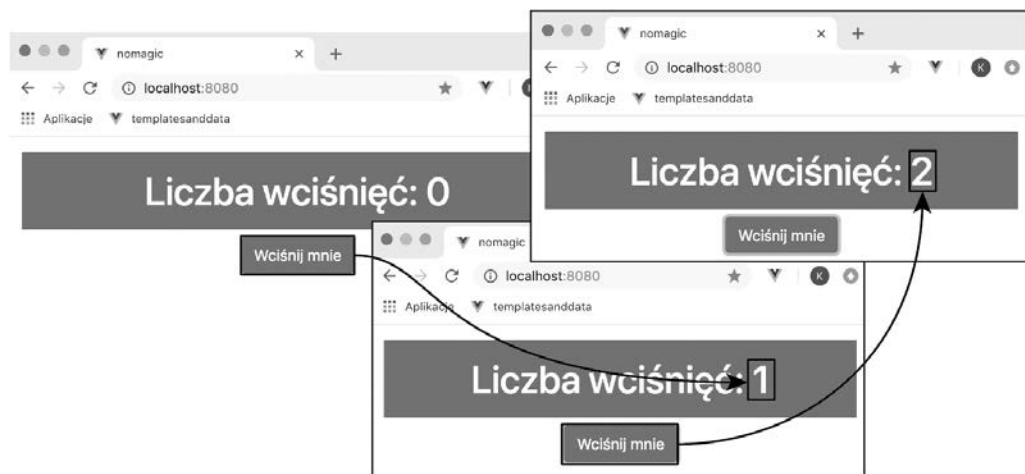
```

    }
  }
});

```

Atrybut, który dodaję do elementu `button`, to **dyrektywa**. Jest to mechanizm Vue.js, który pozwala na rozszerzenie funkcjonalności elementu HTML. W tym przypadku skorzystałem z dyrektywy `v-on`, aby umożliwić obsługę zdarzenia `click` przycisku. Wiążę je z metodą `handleClick`, która została zadeklarowana we właściwości `methods` obiektu konfiguracyjnego `Vue`. Obiekt `methods` definiuje funkcję `handleClick`, która zwiększa wartość zmiennej `counter`.

Skoro Vue.js działa reaktywnie, zmiana wartości zmiennej `counter` pociąga za sobą ponowną ewaluację wiązania danych w elemencie `h1`, a w konsekwencji — zmianę komunikatu wyświetlanego użytkownikowi (rysunek 9.5).



Rysunek 9.5. Obsługa zdarzenia

Modyfikacja komunikatu

Ostatnią zmianą niezbędną do zrównania tego przykładu z poprzednim jest wyświetlenie innego komunikatu, gdy użytkownik nie zdąży jeszcze kliknąć przycisku. Aby osiągnąć ten sam efekt za pomocą obiektu `Vue`, wprowadziłem zmiany w listingu 9.10.

Listing 9.10. Wyświetlanie warunkowego komunikatu w pliku `src/main.js`

```

require('../node_modules/bootstrap/dist/css/bootstrap.min.css')
import Vue from "vue"
new Vue({
  el: "#app",
  template: `<div class="text-center p-3">
    <h1 class="bg-secondary text-white p-3">
      {{ message }}
    </h1>
    <button class="btn btn-secondary" v-on:click="handleClick">
      Wciśnij mnie
    </button>
  </div>`,

```

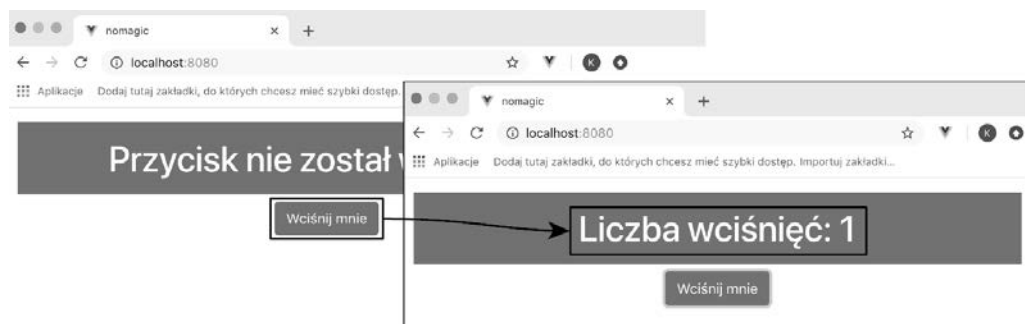
```

data: {
  counter: 0
},
methods: {
  handleClick() {
    this.counter++;
  }
},
computed: {
  message() {
    return this.counter == 0 ?
      "Przycisk nie został wciśnięty" : `Liczba wciśnień: ${this.counter}`;
  }
}
});

```

Właściwość `computed` określa wartości danych, które wymagają dokonania pewnych operacji (często obliczeń), aby uzyskać rezultat. W tym przykładzie korzystamy z wyrażenia, aby sprawdzić wartość właściwości `counter` i na podstawie tego sprawdzenia zwracamy tekst. Nowo obliczona właściwość nazywa się `message` i wyświetlamy jej wartość w elemencie `h1`, umieszczając nazwę w wiązaniu danych, tak jak wcześniej zrobiliśmy to ze zmienną `counter`.

Gdy użytkownik klika przycisk, dyrektywa `v-on` wywołuje metodę `handleClick`, która zwiększa wartość zmiennej `counter`. Vue.js wykrywa zmianę i ponownie ewaluuje wyrażenia wiązań danych zdefiniowane w treści szablonu. Otrzymujemy nową wartość dla obliczonej właściwości, która to wartość jest wyświetlana użytkownikowi za pomocą elementu `h1` (rysunek 9.6).

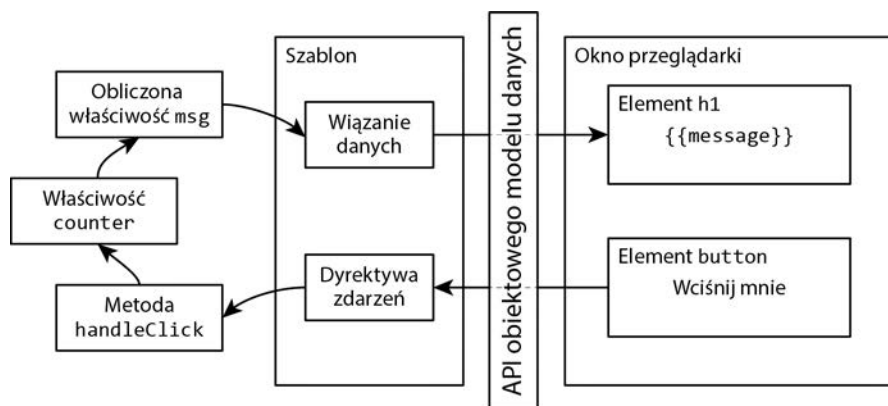


Rysunek 9.6. Zastosowanie właściwości obliczanej

Zasada działania obiektu Vue

Obiekt Vue ma taki sam zakres funkcjonalny jak kod zastosowany w listingu 9.6, jednak dzięki Vue.js aplikacja ma znacznie lepszą strukturę, co pozwala na łatwiejsze zarządzanie i rozwijanie oprogramowania niż w przypadku pracy z modelem DOM. Trudno docenić Vue.js, patrząc tylko na obiekt z listingu 9.10, jednak już w tym przykładzie widać strukturę, w której istnieje podział na treść, logikę i dane aplikacji jak na rysunku 9.7.

Szablon obiektu Vue dostarcza treść HTML, którą widzi użytkownik. Ponadto otrzymujemy wiązanie danych i dyrektywę zdarzenia, która zapewnia interaktywność aplikacji. Dane aplikacji zawierają właściwość `counter` i właściwość obliczoną `msg`, a logika, która łączy je ze sobą, jest zawarta w metodzie `handleClick`.



Rysunek 9.7. Struktura aplikacji

Komponenty w praktyce

Obiekt Vue pomaga oddzielić różne aspekty aplikacji, jednak rezultat wcale nie jest satysfakcjonujący. W większości projektów obiekt Vue jest używany do skonfigurowania aplikacji, ale jej zasadnicze funkcje są definiowane za pomocą **komponentów**, które rozszerzają możliwości obiektu Vue. Ponadto komponenty mogą być definiowane w plikach zawierających użyteczną mieszankę różnego rodzaju treści.

Komponenty definiuje się w plikach z rozszerzeniem `.vue`. Tuż po utworzeniu projekt zawiera plik `App.vue`; w listingu 9.11 przedstawiam jego nową zawartość.

Listing 9.11. Zamiana zawartości pliku `src/App.vue`

```
<script>
export default {
  template: `<div class="text-center p-3">
    <h1 class="bg-secondary text-white p-3">
      {{ message }}
    </h1>
    <button class="btn btn-secondary" v-on:click="handleClick">
      Wciśnij mnie
    </button>
  </div>`,
  data: function () {
    return {
      counter: 0
    }
  },
  methods: {
    handleClick() {
      this.counter++;
    }
  },
  computed: {
    message() {
      return this.counter == 0 ?
        "Przycisk nie został wciśnięty" : `Liczba wciśnień: ${this.counter}`;
    }
  }
}
</script>
```

Komponent przedstawiony w listingu 9.11 zawiera te same funkcje co w przypadku definicji w obiekcie Vue, z wyjątkiem tego, że właściwości języka JavaScript są zdefiniowane w elemencie `script`. Na pierwszy rzut oka może wydawać się, że nie jest to duże usprawnienie, ale jest to dobry przykład na to, że komponenty są obiektami Vue, podobnymi do tego z pliku *main.js*. Komponenty pozwalają na tworzenie aplikacji z wielu komponentów, które można łączyć, aby zapewnić użytkownikowi zaawansowane możliwości.

Kod w języku JavaScript jest zawarty w elemencie `script` z tym samym zakresem funkcjonalnym co obiekt Vue. W porównaniu z wersją ze zwykłym obiektem Vue są dwie zmiany. Pierwsza z nich dotyczy konieczności wyeksportowania obiektu konfiguracji:

```
...
export default {
  // ...właściwości konfiguracji zostały pominięte...
}
...
```

W ten sposób obiekt może zostać zaimportowany ze swojego modułu, co osiągamy w listingu 9.12. Druga zmiana polega na wyrażeniu właściwości `data` w formie funkcji:

```
...
data: function () {
  return {
    counter: 0
  }
},
...
```

Właściwość `data` otrzymuje funkcję, która zwraca obiekt. Właściwości tego obiektu dostarczają wartości danych dla komponentu. Jest to nieco dziwne podejście, ale Vue.js wymaga tego, aby zapewnić, że każdy komponent dysponuje własnymi danymi.

Rejestracja i wdrażanie komponentu

Komponenty należy zarejestrować w Vue.js przed pierwszym użyciem. Z reguły odbywa się to w pliku *main.js*. W listingu 9.12 zamieniam konfigurację obiektu Vue w taki sposób, aby skorzystać z nowego komponentu.

Listing 9.12. Konfiguracja komponentu w pliku *src/main.js*

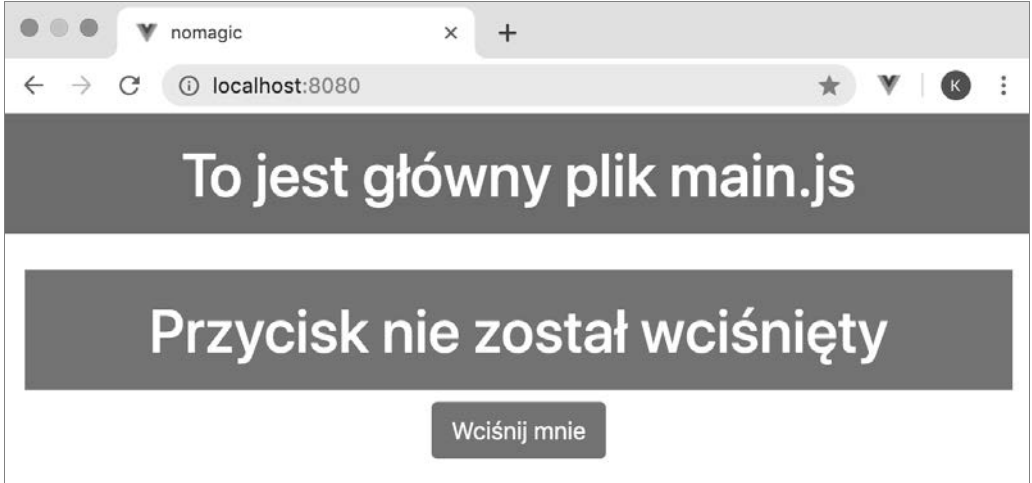
```
require('./node_modules/bootstrap/dist/css/bootstrap.min.css')
import Vue from "vue";
import MyComponent from "./App";
new Vue({
  el: "#app",
  components: { "custom": MyComponent },
  template: `<div class="text-center">
    <h1 class="bg-primary text-white p-3">
      To jest główny plik main.js
    </h1>
    <custom />
  </div>`
});
```

Pierwsza nowa instrukcja korzysta ze słowa kluczowego `import`, które importuje moduł z pliku *App.vue* i nadaje mu nazwę `MyComponent`. Zgodnie z konwencją nazwa pliku powinna być używana jako nazwa importu, ale możesz skorzystać z takiej nazwy, która wydaje Ci się najbardziej sensowna.

Kolejna zmiana polega na dodaniu właściwości `components` do konfiguracji obiektu `Vue`. Ta właściwość powoduje otrzymanie obiektu, który dostarcza do `Vue.js` listę elementów HTML i skojarzonych z nimi komponentów. W tym przypadku skonfigurowałem właściwość `components` tak, aby element `custom` został skojarzony z komponentem `MyComponent`, do którego odwołuję się w pliku `App.vue`.

Zmieniłem także treść szablonu (`template`), dzięki czemu zawiera on prostszy zbiór elementów HTML, wśród których zawarty jest element `custom`.

Gdy obiekt `Vue` podczas przetwarzania swojego szablonu napotka element `custom`, element ten zostanie zastąpiony przez treść zdefiniowaną w ramach komponentu w pliku `App.vue`. W rezultacie użytkownik otrzyma połączenie kodu HTML z właściwości `template` komponentu, a także z właściwości `template` obiektu `Vue`. Po zapisaniu zmian przeglądarka zostanie odświeżona, a Ty zobaczysz treść jak na rysunku 9.8.



Rysunek 9.8. Komponent w praktyce

Obiekt `Vue` zazwyczaj nie wyświetla swojej własnej treści. Zgodnie z konwencją kod HTML umieszcza się w szablonie komponentu, jak w listingu 9.13.

Listing 9.13. Usuwanie treści szablonu w pliku `src/main.js`

```
require('../node_modules/bootstrap/dist/css/bootstrap.min.css')
import Vue from "vue";
import MyComponent from "./App";
new Vue({
  el: "#app",
  components: { "custom": MyComponent },
  template: "<custom />"
});
```

Obiekt `Vue` zastępuje element `div` w pliku `index.html` treścią elementu `custom`, zadeklarowaną w elemencie `template` komponentu. Po zapisaniu zmian zobaczysz kompletny przykład aplikacji, jak na rysunku 9.9.

Oddzielanie szablonu od kodu JavaScript

Szablon komponentu można definiować za pomocą właściwości `template`, jednak warto rozważyć inne podejście, które preferuje wielu programistów. W listingu 9.14 usuwam właściwość `template` i zastępuję ją elementem `template`, zgodnie z typową metodą deklarowania komponentów.



Rysunek 9.9. Kompletna przykładowa aplikacja

Listing 9.14. Zastosowanie elementu `template` w pliku `src/App.vue`

```
<template>
  <div class="text-center p-3">
    <h1 class="bg-secondary text-white p-3">
      {{ message }}
    </h1>
    <button class="btn btn-secondary" v-on:click="handleClick">
      Wciśnij mnie
    </button>
  </div>
</template>
<script>
export default {
  data: function () {
    return {
      counter: 0
    },
    methods: {
      handleClick() {
        this.counter++;
      }
    },
    computed: {
      message() {
        return this.counter == 0 ?
          "Przycisk nie został wciśnięty" : `Liczba wciśnięć: ${this.counter}`;
      }
    }
  }
}
</script>
```

Element `template` zawiera tę samą treść co właściwość `template`, ale to podejście pozwala na traktowanie kodu w większości edytorów jako HTML, co owocuje wsparciem na poziomie edytora, takim jak autouzupełnianie nazw czy oznaczanie błędów. Nic się nie zmienia w wyglądzie i zachowaniu — ta zmiana ma na celu uczynienie procesu deweloperskiego łatwiejszym.

Tworzenie odrębnych plików JavaScript i HTML

Nie każdy lubi łączyć kod JavaScript z HTML-em w tym samym pliku. Jeśli tak jest i w Twoim przypadku, to możesz tworzyć odrębne pliki na oba rodzaje kodu. W listingu 9.15 przedstawiam przykład pliku zawierającego wyłącznie kod HTML.

Listing 9.15. Zawartość pliku *src/App.html*

```

<div class="text-center p-3">
  <h1 class="bg-secondary text-white p-3">
    {{ message }}
  </h1>
  <button class="btn btn-secondary" v-on:click="handleClick">
    Wciśnij mnie
  </button>
</div>

```

W listingu 9.16 usuwam zawartość elementu `template` i dodaję atrybut `src`, który informuje Vue.js, że treść komponentu można znaleźć w odrębnym pliku HTML.

Listing 9.16. Odwołanie do pliku HTML w pliku *src/App.vue*

```

<template src="./App.html" />
<script>
export default {
  data: function () {
    return {
      counter: 0
    }, methods: {
      handleClick() {
        this.counter++;
      }
    },
    computed: {
      message() {
        return this.counter == 0 ?
          "Przycisk nie został wciśnięty" : `Liczba wciśnień: ${this.counter}`;
      }
    }
  }
}
</script>

```

Podsumowanie

W tym rozdziale wyjaśniłem, że Vue.js korzysta z tego samego API modelu DOM do tworzenia aplikacji webowych, które może być używane w czystym, „waniliowym” JavaScriptcie. Za Vue.js nie kryje się bowiem żadna magia. Jak przekonasz się w kolejnych rozdziałach, choć można bezpośrednio korzystać z API modelu DOM, Vue.js zapewnia dużo przydatnych funkcji, które pozwalają na tworzenie bardziej eleganckich, możliwych do zarządzania i skalowalnych funkcji. W kolejnym rozdziale objaśnię strukturę projektów Vue.js i działanie narzędzi deweloperskich.

Skorowidz

A

- administrowanie, 149, 173
- adres URL, 122
- akcje magazynu danych, 467
- aktualizacja, 344, 395
 - treści, 297
- alias, 289
 - trasy, 518
- analiza
 - dokumentu HTML, 56
 - stanu aplikacji, 231
 - zmian w magazynie danych, 460
- Angular, 45
- animacje, 598, 601
 - CSS, 595
- API, Application Programming Interface, 46
 - historii HTML5, 515
 - modelu DOM, 195
- aplikacja
 - Sklep sportowy, 95
- aplikacje
 - debugowanie, 231
 - planowanie układu, 547
 - Vue.js, 209
 - wielostronicowe, 44
- argument własnej dyrektywy, 621
- argumenty, 256
- asynchroniczne
 - komponenty, 494
 - wykonywanie operacji, 91

- atak typu XSS, 435
- atrybuty, 55
 - bez wartości, 56
 - elementu router-link, 539
- automatyczna nawigacja, 490

B

- białe znaki, 345
- biblioteka
 - Axios, 437
 - do obsługi animacji, 598, 601
- błędy, 224, 444
 - komponentów, 400
 - ładowania, 581
- Bootstrap
 - CSS, 194
 - marginesy, 59
 - odstęp, 59
 - stylowanie formularzy, 62
 - stylowanie tabel, 60
 - tworzenie siatki, 60

C

- catchall, 516
- ceny produktów, 106
- checkbox, 33
- CORS, Cross-Origin Resource Sharing, 435
- CSS, Cascading Style Sheets, 28, 51, 58, 272
- cykl życia komponentu, 385, 489

D

debugger, 231
 debugowanie aplikacji, 231
 definiowanie
 reguł walidacji, 353
 strażników tras, 570
 tras, 548
 Docker, 188
 dodawanie
 argumentu, 622
 argumentu filtru, 256
 Bootstrapa, 28
 elementu do tablicy, 155
 filtru, 255
 komponentów, 547
 komponentów administracyjnych, 163
 listy produktów, 106
 nawigacji, 492, 493
 operacji HTTP, 441
 pakietów, 96
 pakietu Bootstrap CSS, 240
 propa, 368
 przycisku wyboru, 33
 strażnika trasy, 166
 stylów CSS, 272
 treści dynamicznych, 29
 wiązań dwukierunkowego, 334
 dołączanie stylów CSS, 97
 DOM, Document Object Model, 46, 195, 281
 domieszki, 626, 627
 domknięcia, 74
 dopasowanie adresów URL, 525, 544
 dostęp
 do gettera, 463
 do komponentu, 572
 do konfiguracji trasowania, 507
 do magazynu danych, 459
 do modelu DOM, 390
 do stanu modułu, 476
 duplikacja zdarzeń, 326
 dynamiczne dopasowywanie tras, 522
 dynamiczne komponenty, 481
 dynamiczne wyświetlanie komponentów, 485
 dyrektywa, 201, 261
 Repeater, 285
 v-bind, 275, 279, 281
 v-else, 270
 v-for, 285, 293, 301–303

 v-if, 272
 v-model, 337, 343, 348
 v-on, 311, 374
 v-text, 264
 v-for
 dyrektywy
 globalne rejestrowanie, 619
 jednofunkcyjne, 625
 modyfikatory, 622
 skrótowe, 275, 315
 tworzenie, 616
 działanie
 aplikacji wielostronicowych, 44
 Bootstrapa, 58
 obiektu Vue, 193, 202
 obietnic, 90

E

edytor, 23
 produktów, 178
 elastyczne formatowanie znaczników, 364
 element, 54
 router-link, 537, 539, 542
 router-view, 124, 553, 556
 script, 243
 slot, 376
 style, 243
 template, 242
 elementy
 formularzy, 331
 kontrola widoczności, 268
 nawigacji, 550, 592
 oznaczanie, 292
 pobieranie indeksu, 293
 powtarzanie, 302
 puste, 55
 trasowanie URL, 535
 typu select, 341
 ukrywanie, 274
 ustawianie właściwości, 282
 wyświetlanie, 267
 zawartość tekstowa, 263

F

faza
 aktualizacji, 392
 celu, 323
 montażu, 390

- przechwytywania, 322
- tworzenia, 389
- zniszczenia, 398
- filtr currency, 131
- filtrowanie, 305
 - zakończonych zadań, 34
- filtry, 255
 - konfiguracja, 256
 - łączenie, 257
- formatowanie
 - adresów URL, 514
 - wartości danych, 255, 343
 - znaczników, 364
- formularze, 62, 331
 - walidacja danych, 141, 351
 - własne wartości, 348
- framework
 - Bootstrap CSS, 194
 - do obsługi stylów CSS, 28
- funkcje, 68, 248
 - administracyjne, 161, 173
 - administracyjne na żądanie, 183
 - haków, 624
 - haków dyrektyw, 618
 - importowanie, 89
 - jako metody, 85
 - obsługi błędów, 403
 - projektu Vue.js, 211
 - strzałkowe, 72
 - Vue.js, 613
 - walidacji, 354
 - z parametrami, 70
 - zaawansowane, 383
 - zmiana nazw, 89
 - zwracające wyniki, 71

G

- getter, 463
- Git, 23
- globalne rejestrowanie dyrektyw, 619

H

- HMR, Hot Module Replacement, 222
- HTML, 51, 53
- HTTP, 221

I

- implementacja
 - edytora produktów, 178
 - funkcji koszyka, 124
 - uwierzytelniania, 161
 - zarządzania zamówieniami, 169
- importowanie
 - modułu, 89
 - pojedynczych funkcji, 90
- indeks
 - elementu, 293
 - w dyrektywie v-for, 295
- instalacja
 - edytora, 23
 - narzędzia Git, 231
 - Node.js, 21
 - pakietu @vue/cli, 22
 - pakietu Bootstrap, 52
 - pakietu HTTP, 429
 - przeglądarki, 24
- instrukcje, 68
 - warunkowe, 77
- interpolacja tekstu, 30, 239
- izolacja komponentów, 366

J

- JavaScript, 65, 205
- jawna konwersja typów, 78
- język HTML, 53

K

- kaskadowe arkusze stylów, *Patrz style CSS*
- katalog dist, 235
 - pakietów, 216
 - productapp, 428
 - src, 215
- klasy
 - aktywnej trasy, 545
 - Bootstrapa, 58
 - kontekstowe, 59
 - przejsć, 596, 599
 - zagnieżdżone, 551
- klawiatura, 328
- klucz, 291

kod

asynchroniczny, 92

HTML, 265

źródłowy, 214

kompilacja, 219

kompilator Babel, 221

komponent ProductEditor, 429

komponent-dziecko, 365, 374

komponent-rodzic, 369

komponenty, 203, 359

administracyjne, 163, 167

asynchroniczne, 494

bez obsługi trasowania, 582

cykl życia, 385, 489

do wyświetlania produktu, 408

do wyświetlania zdarzeń, 446

domieszki, 627

dynamiczne, 481, 485

edytora produktu, 409

faza aktualizacji, 392

faza montażu, 390

faza tworzenia, 389

faza zniszczenia, 398

globalna rejestracja, 365

izolacja, 366

luźno powiązane, 405

mapowanie funkcji, 471

metody cyklu życia, 388

metody ochronne, 570

na żądanie, 577

obsługa błędów, 400, 403

omówienie nazw, 363

paczki, 500

pobieranie danych trasowania, 519

rejestracja, 204

rodzica i dziecka, 363

składniki, 242

sloty, 376

stosowanie propów, 368

strażnik trasy, 570, 577

tworzenie, 589

uwierzytelniania, 182

wdrażanie, 204

z komunikatem ładowania, 578

zmiany, 243

komunikacja⁴¹ między funkcjami haków, 624

komunikat, 201

konfiguracja

dopasowania tras URL, 513

filtrów, 256

klas, 276

komponentu, 204

leniwego ładowania, 498

lintera, 212

narzędzi deweloperskich, 233

segmentu dynamicznego, 523

trasowania URL, 122, 592

usługi, 440

właściwości mode, 515

konsolidacja aktualizacji, 393

konteksty stylu Bootstrapa, 59

kontener Dockera, 188, 189

konwersja

liczb na łańcuchy znaków, 78

łańcuchów znaków na liczby, 79

kopiowanie właściwości, 85

koszyk zakupowy, 121, 124

testowanie funkcji, 132

utrwalanie, 132

widżet podsumowania, 135

wyświetlanie zawartości, 128

L

leniwe ładowanie, 496, 498, 577

liczba elementów, 155

liczby, 76

linter, 212, 226

dostosowywanie reguł, 229

lista

produktów, 104, 175

zadań, 31

literały

obiektowe, 84

tablicowe, 80

tekstowe w atrybutach, 56

lokalizacja aplikacji, 256

Ł

ładowanie komponentów na żądanie, 577

łańcuchy

szablonowe, 75

znaków, 75

łączenie

filtrów, 257

komponentów, 500

plików, 89

propów i zdarzeń, 374

M

magazyn danych, 101, 181, 449
 akcje, 467
 analiza zmian, 460
 mapowanie funkcji, 471
 moduły, 474
 powiadomienia o zmianach, 468
 rejestrowanie modułu, 475
 rozszerzanie, 125, 162
 stosowanie, 456
 tworzenie, 452
 używanie, 452
 Vuex, 456
 właściwości obliczane, 461

magazyn produktów, 103
 mapowanie funkcji, 471
 margines, 59
 mechanizm
 przejścia CSS, 595
 Vue Router, 511

metoda, 252
 beforeRouteEnter, 572
 beforeRouteUpdate, 533
 get, 437

metody
 cyklu życia, 388, 389
 do obsługi tablic, 82
 do obsługi zdarzeń, 313
 HTTP, 433
 nawigacji, 511
 obsługi tablic, 83
 ochronne komponentów, 570
 typu string, 75
 Vue, 633

model DOM, 195

moduły, 86
 importowanie, 89
 łączenie plików, 89
 magazynu danych, 474
 tworzenie, 86
 tworzenie wielu mechanizmów, 88
 używanie, 86
 wieloplikowe, 90

modyfikacja
 komponentu, 156
 komunikatu, 201
 zawartości tablicy, 80

modyfikator
 number, 343
 once, 326
 trim, 345

modyfikatory
 dyrektywy v-model, 343
 obsługi zdarzeń, 320, 325
 własnej dyrektywy, 622
 zdarzeń klawiatury, 328
 zdarzeń myszy, 327

mutacje, 454, 468

mysz, 327

N

narzędzia
 deweloperskie, 25, 218
 projektowe, 100
 Vue.js, 209

narzędzie
 Git, 23
 NPM, 217

nasłuchiwanie zdarzeń, 317

nawigacja, 490
 elementy HTML, 512
 globalny strażnik, 562
 obsługa zmian, 531

nawigowanie do adresów URL, 510

nazwy importowanych funkcji, 89

NPM, 217

O

obiekt
 \$route, 521
 Vue, 198, 199

obiekty, 82
 globalne, 248
 kopiowanie właściwości, 85
 wyliczanie właściwości, 298

obietnica, 90, 437

obserwator, 396, 484

obsługa
 akcji, 156
 animacji, 598, 601
 błędów, 444
 komponentów, 400
 ładowania, 581
 danych, 150

obsługa

- danych odpowiedzi, 435
- dyrektywy Repeater, 285
- elementów formularzy, 331
- elementów router-link, 537
- liczb, 76
- łańcuchów znaków, 75
- mechanizmu wyboru produktów, 126
- nazwanych elementów router-view, 553
- obiektów, 82
- starych adresów URL, 550
- stronicowania listy, 108
- stylów CSS, 28
- tablic, 79
- wartości logicznych, 74
- wyboru kategorii, 114
- wyszukiwania, 157
- zdarzeń, 200, 309–313, 318, 320
- zmian w nawigacji, 531
- zadań HTTP, 434

ochrona tras, 562, 563

odbieranie zdarzeń, 421

odpowiedź HTTP, 437

ograniczenia zawartości elementów, 55

opcje

- konfiguracji leniwego ładowania, 498
- konfiguracyjne, 233
- obserwatora, 398
- reguł lintera, 226

operacje

- asynchroniczne, 91, 464
- HTTP, 444

operator

- identyczności, 77
- rozwinienia, 81
- równości, 77

operatory języka JavaScript, 76

opóźnianie aktualizacji, 344

oznaczanie elementu, 292

P

pakiet

- @vue/cli, 22
- Bootstrap, 194
- HTTP, 429
- json-server, 98, 428
- Vue Router, 508
- Vuex, 450

pakiety

- globalne, 217
- lokalne, 217

parametr reszty, 70

parametry

- domyślne, 70
- metody beforeRouteUpdate, 533

pierwsza aplikacja, 21, 24

planowanie układu aplikacji, 547

pliki, 25

- HTML, 206
- JavaScript, 206

plik

- jsprimer/package.json, 67
- operations.js, 88
- package.json, 99, 218
- productapp/package.json, 406
- productapp/restData.js, 428
- ProductList.vue, 108
- projecttools/package.json, 228, 229, 234
- projecttools/server.js, 236
- projecttools/vue.config.js, 233
- sportsstore/authMiddleware.js, 99
- sportsstore/data.js, 98, 149
- sportsstore/data.json, 185
- sportsstore/deploy-package.json, 188
- sportsstore/server.js, 187
- src/App.html, 207
- src/App.vue, 26, 27, 33, 38, 47, 60, 62, 104, 119, 134, 203, 206, 222, 223, 234, 250, 292, 499, 507, 540, 541, 575, 602
- src/Child.vue, 368
- src/components/admin/Admin.vue, 168
- src/components/admin/Authentication.vue, 164, 183
- src/components/admin/index.js, 183
- src/components/admin/OrderAdmin.vue, 168, 170
- src/components/admin/ProductAdmin.vue, 167, 176
- src/components/admin/ProductEditor.vue, 177, 178
- src/components/App.vue, 495
- src/components/CartSummary.vue, 135
- src/components/CategoryControls.vue, 116, 156
- src/components/Checkout.vue, 138, 143, 144
- src/components/Child.vue, 361, 368, 376
- src/components/DataSummary.vue, 494
- src/components/EditorField.vue, 409, 425

src/components/FilteredData.vue, 572, 581
 src/components/ListMaker.vue, 590, 604
 src/components/ListMakerControls.vue, 591, 607
 src/components/MessageDisplay.vue, 401, 583
 src/components/Numbers.vue, 589, 609, 610, 614, 616, 620, 622, 627, 634
 src/components/OrderThanks.vue, 139
 src/components/PageControls.vue, 113, 151
 src/components/Preferences.vue, 547
 src/components/ProductDisplay.vue, 408, 422, 430, 442, 451, 472, 475, 483, 510, 519, 530, 536
 src/components/ProductEditor.vue, 48, 409, 424, 429, 512, 520, 524, 527, 529, 531
 src/components/ProductList.vue, 105, 107, 111, 126
 src/components/Search.vue, 160
 src/components/ShoppingCart.vue, 129
 src/components/ShoppingCartLine.vue, 128
 src/components/SimpleDisplay.vue, 589, 594, 598, 599
 src/components/Store.vue, 103, 106, 116, 136, 160
 src/components/Subtraction.vue, 628
 src/components/ValidationError.vue, 141
 src/directives/colorize.js, 616, 623, 624
 src/main.js, 28, 69, 97, 102, 131, 199, 220
 src/math/index.js, 91
 src/mixins/numbersMixin.js, 626
 src/plugins/math/componentFeatures.js, 631
 src/plugins/math/directives.js, 630
 src/plugins/math/filters.js, 630
 src/plugins/math/globals.js, 630
 src/plugins/math/Operation.vue, 631
 src/ProductEditor.vue, 416
 src/restDataSource.js, 440
 src/router/basicRoutes.js, 561, 575, 577, 580, 583
 src/router/index.js, 122, 140, 165, 166, 168, 177, 505, 518, 519, 522, 525, 526, 548, 555, 562, 563, 564, 568, 592
 src/store/auth.js, 162, 182
 src/store/cart.js, 125, 133
 src/store/index.js, 49, 102, 108, 115, 118, 125, 138, 153, 157, 174, 182, 453, 465, 490, 578
 src/store/orders.js, 137, 169
 src/store/preferences.js, 478
 src/validationRules.js, 353
 templatesanddata/package.json, 240

pobieranie danych, 439
 aplikacji, 483
 trasowania, 519
 pochylenie, 595
 odpowiedź wstępnego pobierania, 497
 polecenia narzędzia NPM, 217
 polimorfizm, 70
 powiadomienia o zmianach, 468
 powtarzanie
 elementów, 302
 treści, 302
 poziomy reguł lintera, 227
 projekt
 Sklep sportowy, 95
 Vue.js, 211, 214
 prop, 368, 372
 propagacja zdarzeń, 320, 324
 protokół CORS, 435
 przechowywanie danych, 38, 39
 przeglądanka, 24
 przejścia, 587, 593, 603
 dla zmian w kolekcji, 604
 początkowe, 597
 przekazywanie
 argumentów, 464
 funkcji przez argument, 71
 propa, 585
 przekierowanie, 565
 do trasy nazwanej, 566
 żądania, 564
 przesłanianie usług, 413
 przestrzeń nazw modułów, 478
 przesunięcie, 595
 przetwarzanie danych, 438
 przycisk wyboru, 33

R

React, 45
 reagowanie na zmiany, 357, 609
 reaktywność, 250, 390
 reguły
 lintera, 229
 walidacji, 353
 rejestrowanie
 dyrektyw, 619
 komponentów, 138, 204, 365
 modułu, 475
 wtyczek, 633

renderowanie po stronie serwera, 45
 REST, Representational State Transfer, 98, 433
 restartowanie aplikacji, 29
 REST-owa usługa sieciowa, 117, 162, 427, 433
 routing, 122, 503
 rozszerzanie
 możliwości Vue.js, 613
 magazynu danych, 125
 Vue Devtools, 231

S

segmenty
 dynamiczne, 526
 opcjonalne, 526
 sekwencja przejścia, 597
 selektor CSS, 58, 273
 serwer
 deweloperski HTTP, 221
 HTTP, 236, 428
 siatka, 60
 skalowanie, 149, 595
 Sklep sportowy, 95
 administrowanie, 149, 173
 dodawanie
 funkcji administracyjnych, 173
 pakietów, 96
 zamówień, 137
 edytor produktów, 178
 formularz walidacji, 141
 funkcje administracyjne, 161
 koszyk zakupowy, 121
 lista produktów, 104, 106
 magazyn danych, 101
 magazyn produktów, 103
 obsługa
 danych, 150
 rozliczenia, 137
 wyszukiwania, 157
 przetwarzanie cen, 106
 REST-owa usługa sieciowa, 98, 117
 rozliczenie, 121
 skalowanie, 149
 stronicowanie listy produktów, 108
 style CSS, 97
 uruchamianie aplikacji, 101
 wdrażanie, 181, 185, 188
 wybór kategorii, 114
 wybór produktów, 126
 wyświetlanie zawartości koszyka, 128

zamówienia, 121
 zarządzanie zamówieniami, 169
 sloty
 nazwane, 377
 o ograniczonym zasięgu, 378
 słowo kluczowe
 async, 92
 await, 92
 const, 73
 debugger, 234
 let, 72
 new, 506
 return, 245
 sortowanie, 305
 SPA, 44
 SSR, server-side rendering, 45
 stan, 454, 468
 stosowanie
 aliasu trasy, 289, 518
 argumentów własnej dyrektywy, 621
 biblioteki do obsługi animacji, 598
 domieszki, 627
 funkcji, 68
 funkcji walidacji, 354
 komponentów, 359
 komponentów asynchronicznych, 494
 lintera, 226
 magazynu danych, 449, 456
 modułów, 86
 modułów magazynu danych, 474, 475
 modyfikatorów
 dyrektywy v-model, 343
 obsługi zdarzeń, 320
 własnej dyrektywy, 622
 obietnic, 91
 propa, 369
 propów, 368
 przejścia, 593, 601–604
 przestrzeni nazw modułów, 478
 REST-owych usług, 427
 slotów komponentów, 376
 slotów nazwanych, 377
 slotów o ograniczonym zasięgu, 378
 systemu trasowania, 507
 właściwości obliczanych, 303
 wtyczki, 633
 wyrażeń regularnych, 525
 wyrażeń własnych dyrektyw, 620
 zdarzeń początkowych, 608
 zdarzeń przejść, 606

- strażnik
 - nawigacji, 562
 - trasy, 166, 577
 - stronicowanie, 151
 - danych, 303
 - struktura
 - dyrektywy v-bind, 275
 - dyrektywy v-on, 311
 - podkatalogów, 24
 - projektu, 213
 - style
 - Bootstrapa, 58
 - CSS, 28, 272
 - stylowanie
 - elementów
 - HTML, 29
 - łącza routera, 542
 - nawigacji, 543
 - formularzy, 62
 - tabel, 60
 - szablon, 205
 - szyna zdarzeń, 420, 421
 - lokalna, 424
- ## Ś
- środowisko programistyczne, 21, 46
- ## T
- tabele
 - naprzemienne kolorowanie wierszy, 296
 - tablice, 79, 287
 - modyfikacja zawartości, 80
 - odczyt, 80
 - przeglądanie zawartości, 81
 - stron, 155
 - wbudowane metody, 82
 - wykrywanie zmian, 296
 - zamiana obiektu, 298
 - testowanie
 - aplikacji, 186
 - funkcji koszyka, 132
 - klas zagnieżdżonych, 551
 - usługi sieciowej, 100
 - transformacja, 219
 - CSS, 595
 - trasowanie
 - do wyświetlania komponentów, 507
 - URL, 122, 503, 535, 601
 - konfiguracja, 592
 - zaawansowane, 559
 - trasy
 - definiowanie, 548
 - dynamiczne dopasowywanie, 522
 - nazwane, 528, 566
 - ochrona, 562
 - typu catchall, 517
 - zagnieżdżone, 546
 - treści
 - dynamiczne, 29
 - zastępcze edytora, 177
 - trwałe przechowywanie danych, 38
 - tworzenie
 - domieszek komponentów, 626
 - dwukierunkowych wiązań modeli, 333
 - elementów nawigacji, 550, 592
 - funkcji, 68
 - globalnego filtru, 131
 - komponentów, 138, 589
 - komponentów rozliczenia, 138
 - kontenera Dockera, 188, 189
 - listy produktów, 104
 - lokalnych szyn zdarzeń, 424
 - magazynu danych, 101, 452
 - magazynu produktów, 103
 - modułów, 86
 - obiektów komponentów, 390
 - obiektu Vue, 198
 - pliku danych, 185
 - pliku paczki, 188
 - projektu, 24, 210
 - projektu Sklep sportowy, 95
 - reaktywnych usług, 415
 - REST-owej usługi sieciowej, 98
 - siatki, 60
 - tras nazwanych, 528
 - tras zagnieżdżonych, 546
 - trasy typu catchall, 517
 - usługi, 411, 419
 - usługi HTTP, 440
 - usługi obsługi błędów, 444
 - wersji produkcyjnej, 234
 - wiązania, 334
 - wiązania do tablicy, 346

tworzenie

- wierszy tabeli, 288
- własnych dyrektyw, 616
- własnych zdarzeń, 373
- wtyczki, 632
- wtyczki Vue.js, 629
- zadań, 36

typy, 72

- prymitywne, 74

U

układ aplikacji, 547

ukrywanie elementów, 274, 582

upraszczanie wiązań dwukierunkowych, 337

URL

- trasowanie, 503

uruchamianie

- aplikacji, 101, 189
- REST-owego serwera, 432

usługa, 411

- HTTP, 440
- obsługi błędów, 444
- reaktywna, 415
- sieciowa, 98
- szyny zdarzeń, 420

ustawianie

- atrybutów, 279, 280
- pojedynczych stylów, 277
- właściwości elementu, 282
- właściwości HTMLElement, 281

usuwanie białych znaków, 345

uwierzytelnianie, 161, 182

V

Vue Router, 511

Vue.js, 19, 43

W

walidacja, 141, 144, 351–354

wartości logiczne, 74

wdrażanie, 185, 233

- aplikacji, 188
- komponentu, 204
- sklepu sportowego, 181
- wersje pakietów, 216

wiązania

- do elementów typu select, 341
- do pól tekstowych, 338
- do przycisków, 339
- do różnych typów danych, 346
- do tablicy, 346
- dwukierunkowe, 333
- z elementami formularzy, 338
- wiązanie danych, 239, 247, 248
- wybór komponentów, 486
- wywoływanie metod, 254

widżet, 135

własne

- dyrektywy, 616, 622
- elementy HTML, 371
- wartości dla przycisków, 349
- wartości formularza, 348
- zdarzenia, 373

właściwości, 58

- elementów, 283
- HTMLElement, 281
- obiektów zdarzeń, 313
- obiektu, 298, 300
- \$route, 521
- walidacji, 142
- obliczane, 249, 250, 251, 303
- odpowiedzi Axios, 438
- w magazynie danych, 461
- zdefiniowane w obiekcie wiązania, 619

wstrzykiwanie zależności, 405, 411, 412, 417

wtyczki, 629, 632, 633

wybór

- fragmentów zawartości, 270
- kategorii, 114
- komponentów, 486
- produktów, 126
- tablicy elementów, 346
- wyświetlanych elementów, 272
- zdarzenia nawigacji, 541

wyliczanie właściwości obiektu, 298

wyłączanie odpowiedzi, 497

wyrażenia

- lambda, 72
- regularne, 525
- złożone, 247

wysyłanie zdarzeń, 420

wyszukiwanie produktów, 157

wyświetlanie
 błędów, 224, 447
 kodu HTML, 265
 komponentów, 485, 507
 komponentów-dzieci, 410
 listy zadań, 31
 produktu, 408
 trasowanego komponentu, 123
 wartości danych, 244
 wybranych elementów, 267, 268
 zawartości koszyka, 128
 zdarzeń, 446
 komponentu, 487
 wartości danych, 246
 wywołanie zwrotne, 395
 wywoływanie metod, 254

X

XSS, Cross-Site Scripting, 435

Z

zadania
 tworzenie, 36
 zakończone, 34
 zamiana treści zastępczych, 26
 zamówienia, 169
 zarządzanie
 propagacją zdarzeń, 320
 zamówieniami, 169, 172
 zdarzeniami obserwatora, 484
 zasada działania
 SPA, 44
 dyrektyw, 618
 obietnic, 90

zasięg CSS, 367
 zawartość tekstowa elementu, 263
 zdarzenia, 200, 309, 312
 duplikacja, 326
 faza celu, 323
 faza przechwytywania, 322
 klawiatury, 328
 metody, 313
 modyfikatory, 320, 325
 myszy, 327
 nasłuchiwanie, 317
 nawigacji, 541
 obserwatora, 484
 obsługa, 311
 odbieranie, 421
 propagacja, 320, 324
 przejść, 606
 tworzenie, 373
 właściwości obiektów, 313
 wysyłanie, 420
 wyświetlanie, 446
 złożoność aplikacji, 46
 zmiana
 adresu URL, 509
 rozmiaru strony, 113
 strażnika, 568
 treści, 111
 zmienne, 72
 jako właściwości obiektów, 84

Ż

żądania HTTP, 221, 428, 436

PROGRAM PARTNERSKI

— GRUPY HELION —

- 
1. ZAREJESTRUJ SIĘ
 2. PREZENTUJ KSIĄŻKI
 3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA
Helion 

Vue.js 2: zyskaj większe możliwości i pisz najlepsze aplikacje!

Vue.js jest frameworkiem, który służy do budowy nowoczesnych, reaktywnych i skalowanych aplikacji WWW, a przy tym ułatwia korzystanie z komponentów wielokrotnego użytku. Framework ten powstał jako narzędzie do szybkiego prototypowania, a teraz dynamicznie się rozwija i ewoluje, dzięki czemu liczba dostępnych funkcji stale rośnie. Prostocie i wszechstronności zawdzięcza ogromne uznanie deweloperów. Z całą pewnością można go uznać za narzędzie niezbędne każdemu, kto buduje kompleksowe aplikacje WWW i pragnie utrzymywać wysokie standardy.

Dzięki tej książce dowiesz się, czym jest Vue.js i jak rozpocząć z nim pracę, przekonasz się także, jakie ma możliwości. Nauczysz się budować dynamiczne aplikacje wykorzystujące właściwości nowoczesnych przeglądarek internetowych i urządzeń. Odkryjesz zalety wzorca MVC (model – widok – kontroler) i dowiesz się, jak zadziwiająco sprawnym językiem stał się JavaScript. Zapoznasz się z anatomią projektu Vue.js, z procesami kompilacji i transformacji oraz nabierzesz biegłości w posługiwaniu się zaawansowanymi funkcjami frameworka. Liczne przykłady pozwolą Ci na niemal natychmiastowe przetestowanie opisywanych zagadnień w praktyce. W rezultacie bardzo szybko będziesz przygotowany do tworzenia zaawansowanych, reaktywnych i dynamicznych aplikacji WWW!

W tej książce między innymi:

- zarys budowy frameworka, jego instalacja i sposoby wykorzystania
- wzorzec MVC i architektura aplikacji w Vue.js 2
- dynamiczne aplikacje WWW po stronie klienta
- korzystanie z usług REST
- rozszerzanie i modyfikowanie Vue.js w zależności od potrzeb

Adam Freeman — jest doświadczonym specjalistą IT. Kierował zespołami inżynierów w wielu firmach, ostatnio jako dyrektor ds. technologicznych i dyrektor ds. operacyjnych w międzynarodowym banku. Po przejściu na emeryturę poświęca czas na pisanie znakomitych książek i biegi długodystansowe.

  helion.pl  HELION SA ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	<i>Sprawdź nasze szkolenia!</i> SZKOLENIA  AKADEMIA IT & BUSINESS WWW.SZKOLENIA.HELION.PL	KOD KORZYŚCI Sięgnij po więcej! ▶  ISBN 978-83-283-5498-2  9 788328 354982 Cena: 99,00 zł
--	---	--

INFORMATYKA W NAJLEPSZYM WYDANIU

Apress®